LBNL DHCP "Jailing" Overview

Mark T. Dedlow

January 2004
LBID-2498

Objective

The goal of the DHCP jailing mechanism is to alter or disable a given host's normal network functionality as a means to A) prevent the host from acting as an infection vector (e.g. for a worm), and/or B) force the (sometimes unknown) owner/user of the host to take some remedial action, for example, applying a patch. These two objectives are sometimes, but not always, related. For example, in the case of a host known to be infected with a worm, we both wish to isolate the host to prevent it from spreading the worm, as well as force the user to repair the system. However, we also jail uninfected hosts in order to force the prophylactic application of patches or reconfiguration of the system.

Basis of operation

A host configured to use DHCP must contact a DHCP server at boot time in order to obtain the requisite network configuration information, including at a minimum the host ip address, subnet mask, and gateway ip address; in addition, hosts typically obtain the address of DNS servers via DHCP as well. In today's heavily network dependent environment, a typical host is severely disabled without networking capability (eg. the user cannot login to central file servers, browser the web, get email, etc). Since the networking capability of a host using DHCP is dependent upon the network configuration provided by the server, a certain (though limited) degree of centralized control can be exercised over each DHCP host by selectively controlling what DHCP configuration information is handed out to clients.

Control options

There are three main DHCP configuration/control options we use, described below. Note that these options are based on the operation of the ISC DHCP server that LBNL uses, although other DHCP servers probably support similar options.

- *Deny DHCP service completely*. The DHCP server can simply refuse to provide service to a given host. Without DHCP service, the exact behavior of the host is operating system dependent, but a common behavior (eg. MS Windows and MacOS) is for the client to auto-configure an ip address in a subnet that will render the host largely inoperable (see Limitations section below for details). In the ISC DHCP server, this behavior (service denial) is termed "deny boot", which is somewhat misleading in that it could seem to imply that the hosts will not boot, which is not generally true (it would be true only in the case where DHCP is used to provide the host a boot image, a feature not used at LBL). Nevertheless, we use the term "deny boot" to describe this host treatment.

- *Provide no default route (gateway address) and one source route*. With this configuration, the host's ability to communicate outside its own subnet is limited to a single destination ip address provided by the DHCP server. In effect, the user will generally perceive this as equivalent to the above, ie. "broken network". The big advantage over the above option is that the user can in fact access a (single) central server from which we can distribute information to the user and patches (for example from a web server, although the server could run any service.) In principle, it should be possible to provide multiple source routes, however, in practice, the most common client type (MS Windows) will only recognize one. We refer to this as "host isolation."

- *In addition to above, provide a "special" DNS server*. When the client host has only a source route to a single server, DNS would normally be disrupted (unless the client were on the same subnet as the DNS server) because the client would not have a route to the DNS server(s). However, the source routed ip address can also be provided as the DNS server address, which is what we do at LBNL. However, the DNS system running on this machine is configured to resolve all hostnames back to itself. The effect on the client host is that every (hostname-based) IP communication attempt gets directed to our special administrative server (which again, happens to be the only machine the client host can successfully communicate with) For most communications this is useless to the user, with one important exception: when the user tries to use the web, they get directed to our administrative web server, no matter what URL they attempt to access. This web server displays a prominent message explaining to the user that their machine is disabled, why, how to fix it, and whom to contact for help if needed.

  Given that the average user of a DHCP host does not have a sophisticated understanding of computer networking, and that a browser is one of the first things most people use, this combination of configurations results in the typical user of a DHCP jailed host self-identifying their predicament within a few minutes of starting up their computer. With appropriately context-specific information presented to the user (ie. the exact reason the specific host was jailed) and problem remediation tools made available (eg. patches and installation instructions on the same website), our preliminary results show that perhaps two thirds of users will self-diagnose their predicament (ie. get to the website and read what's on the screen) and self-remediate (eg. apply patch). Finally, if the website also provides the user with the ability to release themselves from jail, a full self-servicing loop can be implemented.

Note that this special use of the DHCP DNS server option is technically and functionally independent of the above source route option. Even without the DNS trickery, the client host can access our administrative server – and only that server – but must get to it via ip address directly. For a client that has been configured to ignore DHCP DNS information and instead use hard-wired DNS servers, the only difference is how, and importantly how quickly and easily, the user gets to our special server. In most cases, the user finds their machine inoperable and calls our Help Desk, who can then instruct them to enter the ip address of our server into their browser.

Advantages of system : cheap and easy , minimal complexity, no affect on other systems; self-service.

Limitations

The system has a number of constraints that limit its scope and effectiveness.

- *Scope of the hosts affected*. The system, by definition, applies only to DHCP hosts. This is accordingly not a limitation of the DHCP jail system, per se, but could be perceived as a limitation in the larger context of a desire to control all hosts, not just those using DHCP.
- *Local subnet access*. A jailed host can still communicate on the local subnet to varying degrees depending upon the type of jailing. Thus it is not a complete isolation system and cannot fully protect other hosts on the same subnet from certain types of malicious behavior.
- *Latency between configuration and effect*. Since DHCP is a client-initiated protocol, the configuration for a given host only takes effect when that host next performs a DHCP request. This is either at client boot time, or else typically every ½-of-lease-duration for a host that is continually on. At LBL, default lease duration is currently four hours, which is relatively short by most standards (24 hours might be more typical). Accordingly, this is not an *immediate* isolation system, however, note that 1-2 hours is certainly competitive with the time to effect other currently available isolation means (network tracing to identify the location of the host and visiting it). Also, the latency is not an issue for routine policy enforcement on non-infected systems. By and large, we have not found the latency to be a significant drawback, although the idea of instant host isolation remains attractive ideal benchmark, but would be vastly more complex and expensive to achieve in practice.

- *Defeatability*. It would not be difficult for a knowledgeable user to defeat the system, principally by either configuring their system with a fixed IP address or changing their system's MAC address (either via NIC swap or software control). However, in practice this is not a significant drawback, for several reasons, not the least of which is the fact that the system was not intended to defeat malicious intent in the first place. In addition, intentionally defeating the system will only postpone (and aggravate) the client's ultimate reckoning, as they will get ultimately isolated by some other mechanism.

DHCP server configuration

The ISC DHCP server has various features to allow a degree of programmatic control over the treatment of lease requests from specific hosts, which we utilize to implement the jailing system. (Note that there is more than one way to achieve the desired effect, and we only describe our current (beta) implementation. When and if we implement in production, significant changes from this implementation are likely.) The two key configuration elements are 1) to identify specific hosts targeted for jailing by their MAC address within the DHCP server, and 2) to configure the server with conditional logic to treat each client as a member of one of three classes: *normal* (no special treatment), *deny-boot* (deny DHCP service), and *isolated* (no gateway, source route and bogus DNS).

1) *Identifying hosts*
   A specific host can be identified in the DHCP configuration file by a *host-declaration.* This is simply a configuration construct that declares a given MAC address and associates a unique, arbitrary logical name with it. This declaration in and of itself does not affect the server's behavior, but allows other conditional constructs to reference this declaration in several ways. One way is that all clients making a request to the server are classified as either *known* or *unknown* in the server, simply based on whether the host's MAC address has been declared in the configuration. We use this to divide our hosts into two main logical categories with respect to jailing: 1) unknown hosts are normal and not jailed, and 2) known hosts are jailed, with either deny-boot or isolate jail treatment.

   We name our host declarations with the client's MAC address and a label that identifies which jail class they belong to. Other server logic will use this label to differentiate the two classes. Our host declarations look like this:

```
host denyboot-00-50-04-a6-fd-eb { hardware ethernet 00:50:04:a6:fd:eb;
host isolated-00-b0-d0-73-ff-80 { hardware ethernet 00:b0:d0:73:ff:80;
```

2) *Conditional treatment of known hosts*
   When a DHCP request is being processed by the server, there are programmatic statements that can be applied to the current request. The configuration logic and code is very simple and largely self-explanatory: (note that at this stage, the server has already identified the current request as from either a known or unknown host).

```
    subnet 128.3.20.0 netmask 255.255.252.0 {
        # generic config for normal hosts
        option routers 128.3.20.1;
        option subnet-mask 255.255.252.0;
        range 128.3.22.1 128.3.22.254;
        range 128.3.23.1 128.3.23.254;

        # "jailed" handling conditions
        if known {
            if substring(host-decl-name,0,8) = "isolated" {
                option routers 0.0.0.0;
                option static-routes 131.243.129.107  128.3.20.1;
            }
            elsif substring(host-decl-name,0,8) = "denyboot" {
                deny booting;
```

```
                    }
                }
            }
```

Operation

The above describes the concepts and the required DHCP software configuration to jail a host, but in actual operation, if the system is to be efficient, consistent, and reliable it requires some functional encapsulation and automation at various points. In this regard we have developed a unix command-line interface to jailing and unjailing a host, and an automated batch-oriented DHCP configuration editor.

The command line program invoked as follows:

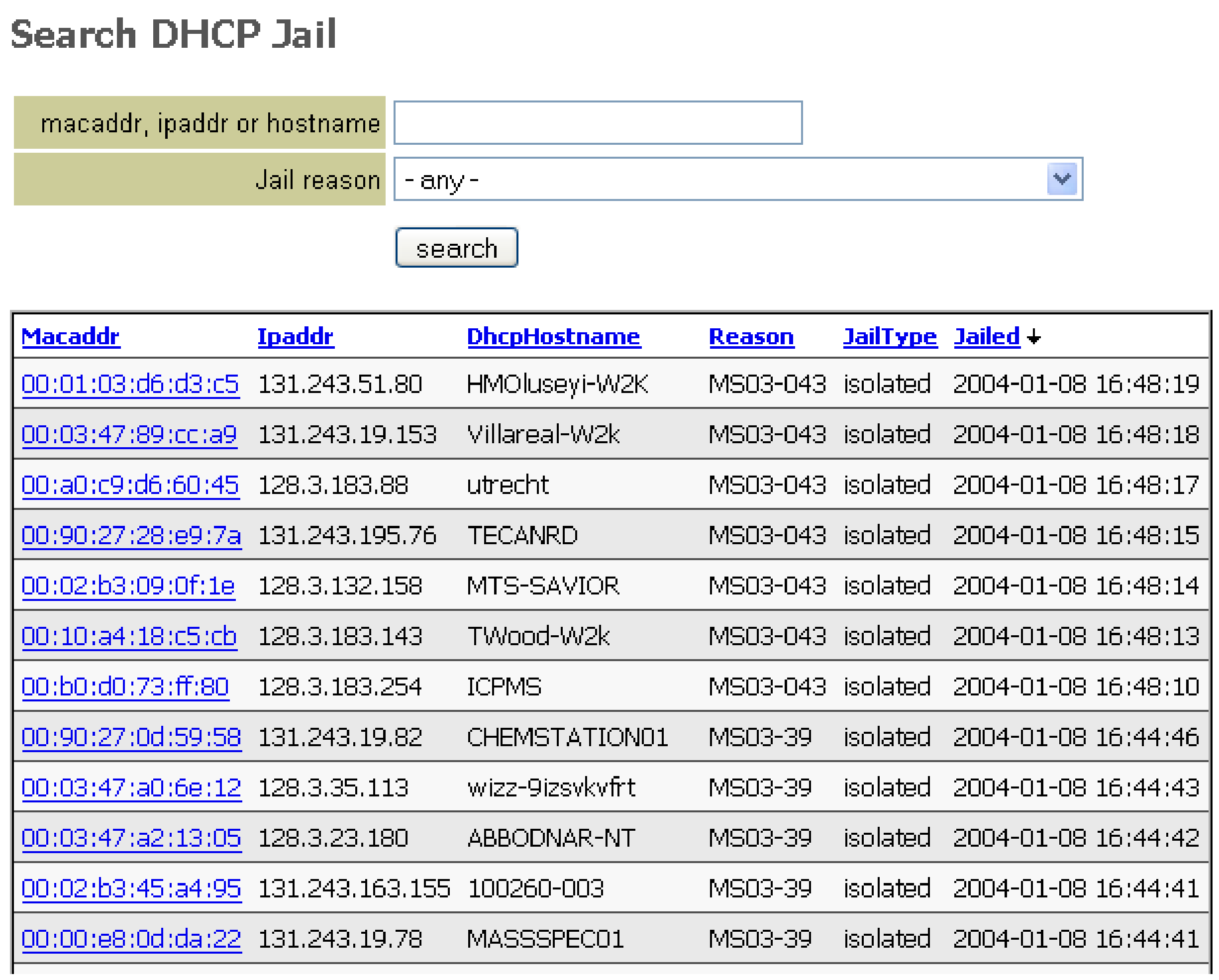
```
% dhcp_jailer -isolate <ipaddress>
% dhcp_jailer -denyboot <ipaddress>
% dhcp_jailer -free <ipaddress>
```

The (perl) source code is attached for reference.

The DHCP server needs to be restarted each time host declarations are added or removed and in order to avoid thrashing the server and to serialize the editing of the configuration file, we implemented a rudimentary jail-request queuing and configuration file editing system. The `dhcp_jailer` command above does not directly edit the DHCP server configuration file, but instead touches a file in a queue directory, where that file represents a jail request for a specific host. Operating out of cron, a separate program, `dhcp_jailer_daemon`, periodically (currently every 10 minutes) processes those requests, editing the DHCP server configuration file and restarting the server. Source code for this is attached.

Additional components

In actual operation, the system's overall utility is operationally enhanced with the addition of a few web pages where appropriate people (eg. Computer Protection and Help Desk staff) can see what hosts have been isolated. For example, here is a screen shot of the web page for looking up and viewing jailed hosts.



macaddr, ipaddr or hostname	
Jail reason	- any -

Macaddr	Ipaddr	DhcpHostname	Reason	JailType	Jailed ↓
00:01:03:d6:d3:c5	131.243.51.80	HMOluseyi-W2K	MS03-043	isolated	2004-01-08 16:48:19
00:03:47:89:cc:a9	131.243.19.153	Villareal-W2k	MS03-043	isolated	2004-01-08 16:48:18
00:a0:c9:d6:60:45	128.3.183.88	utrecht	MS03-043	isolated	2004-01-08 16:48:17
00:90:27:28:e9:7a	131.243.195.76	TECANRD	MS03-043	isolated	2004-01-08 16:48:15
00:02:b3:09:0f:1e	128.3.132.158	MTS-SAVIOR	MS03-043	isolated	2004-01-08 16:48:14
00:10:a4:18:c5:cb	128.3.183.143	TWood-W2k	MS03-043	isolated	2004-01-08 16:48:13
00:b0:d0:73:ff:80	128.3.183.254	ICPMS	MS03-043	isolated	2004-01-08 16:48:10
00:90:27:0d:59:58	131.243.19.82	CHEMSTATION01	MS03-39	isolated	2004-01-08 16:44:46
00:03:47:a0:6e:12	128.3.35.113	wizz-9izsvkvfrt	MS03-39	isolated	2004-01-08 16:44:43
00:03:47:a2:13:05	128.3.23.180	ABBODNAR-NT	MS03-39	isolated	2004-01-08 16:44:42
00:02:b3:45:a4:95	131.243.163.155	100260-003	MS03-39	isolated	2004-01-08 16:44:41
00:00:e8:0d:da:22	131.243.19.78	MASSSPEC01	MS03-39	isolated	2004-01-08 16:44:41

Status of the system at LBNL

We have been using this system in a sort of "beta" mode, testing its utility and reliability for approximately three months. We anticipate incorporating a variation of it into production operations in the near future.

Appendix A: dhcp_jailer source

unix command-line interface to jailing a host

```
#!/usr/bin/perl -w

use strict;
use Getopt::Long;
use File::Basename;

my $program = basename($0);
my $action;
my $host;
my $keyfile = "~dedlow/.ssh/id_rsa_dhcp";
my $remote_user = "capgrbell";
my $mac = '[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}';
my $ip = '\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}';
my $test;


#
# Main
#
$ DHCP_SERVER_HOST = '';

# process command line arguments
GetOptions(
        'isolated=s'           => sub { ($action,$host) = @_ },
        'release|free=s'       => sub { ($action,$host) = @_ },
        'denyboot=s'           => sub { ($action,$host) = @_ },
        'test'                 => \$test,
        'help'         => \&usage,
) or usage(1);

# and require a host argument
usage(1) unless $host;


# validate the host spec
die "$program: invalid host spec: $host\n"
unless $host =~ /^($mac|$ip|$ip\@.+)$/;

# validate IP's are dhcp ips
if ($host =~ /^($ip)$/ or $host =~ /^($ip)\@.+$/ ) {
        die "not a dhcp IP address: $1" unless is_dhcp_ip($1);
}

# exclude VPN
die "you can't modify secure-local: $host\n" if $host =~ /131.243.220.15|00:03:a0:88:d3:09/;


# format request key
my $request = "$action+$host+$ENV{LOGNAME}";


# assemable request command
my $command;
if (`hostname` =~ /^$DHCP_SERVER_HOST/) {
        printf "running local\n";
        $command = "touch ~capgrbell/dhcp-jail-requests/$request";
} else {
        $command = "ssh -i $keyfile -l $remote_user pop './touch ./dhcp-jail-requests/$request'";
}


if ($test) {
        print "The command is: $command\n";
        exit;
}
```

```
# send request

`$command`;
if ($?==0) {
        print "request submitted successfully\n";
        exit 0;
} else {
        print STDERR "ERROR: request filed : $!";
        exit 1;
}



##################################################################
# subs


sub is_dhcp_ip {
        my $ip = shift or die "ip: $ip";
        my $shouldbe = join('.', reverse( split /\./, $ip)) . ".in-addr.dhcp.lbl.gov.";
        chomp(my $dig = `dig -x $ip +short`);

        return $dig =~ /$shouldbe/ ? 1 : undef;
}


sub usage {

print <<EOF;

Usage: $program -isolate | -denyboot < ip | ip\@time | mac >
       $program -free <mac>


$program submits a request to DHCP "jail" (or free from jail) a host.

Go-to-jail requests accept a host specification in one of three forms:

  1) IP addr -- the host that most recently used this address will be jailed.
  2) IP addr + time -- the host that used this address at <time> will be jailed.
  3) MAC address -- this MAC addrress

  A time spec can be unix integer time or most common string formats such as
  syslog time "Sep 17 06:00:00" (be sure to quote if it has whietespace).

Free-from-jail requests require a MAC address.

Examples:

        $program -isolate 131.243.223.168

        $program -isolate 131.243.223.162\@1063614634

        $program -denyboot "131.243.223.162\@Sep 17 06:00:00"

        $program -denyboot "131.243.223.168\@Sep 17 06:00:00"

        $program -free 00:35:60:a2:3:00

EOF

exit shift;

}
```

Appendix B: dhcp_jailer_daemon source

```
Dhcp configuration file batch editor

#!/usr/bin/perl

use strict;
use lib glob '~dedlow/perl/lib';
use Date::Parse;
use Getopt::Long;
use File::Basename;
require "dhcp_jailer.lib";

# gloabl vars
use vars qw($LEASE_FILE $DHCPD_CONF $LOG_FILE $OUTPUT_FILE $JAIL_DB %EMAIL
                      $DEFAULT_EMAIL_NOTIFY
                      $user $log $host $debug $program $force $action $verbose %pat);


# log protos
sub logsay (@);
sub logerr (@);
sub logdie (@);
sub notify (@);


# map username (from queue file) to email addrs
%EMAIL = (     jmel           => 'jmellander@lbl.gov',
                      jekrous        => 'jekrous@lbl.gov',
                      grbell         => 'grbell@lbl.gov',
                      dedlow         => 'mtdedlow@lbl.gov',
                      apache         => 'jekrous@lbl.gov', );

$program = basename($0, '.pl');


# main configuration parameters
$DEFAULT_EMAIL_NOTIFY = 'jekrous@lbl.gov';
$DHCPD_CONF  = "/tftpboot/etc/dhcpd.conf";
$OUTPUT_FILE = $DHCPD_CONF;
$LEASE_FILE  = "/var/db/dhcpd.leases";
$LOG_FILE    = "jailer.log";


# basic patterns
%pat = (
       mac     => '[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-f]{1,2}:[0-9a-
f]{1,2}',
       ip      => '\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}',
);


GetOptions( 'isolated'        => sub { $action = 'isolated' },
                      'denyboot'             => sub { $action = 'denyboot' },
                      'release|free' => sub { $action = 'release' },
                      'host=s'               => \$host,
                      'verbose'              => \$verbose,
                      'requestor=s'  => \$user,
                      'help'                 => \&usage,
                      'conf=s'               => \$DHCPD_CONF,
                      'force'                => \$force,
                      'debug'                => \$debug,
                      'output=s'             => \$OUTPUT_FILE,
                      'ip2mac'               => \&resolve_ip,
                      'log=s'                => \$log,
) or exit 1;


$host ||= shift; # accept host as option argument or as regular argument ARGV[0]
usage() unless $action and $host;
```

```
open(LOG, ">>$LOG_FILE") or die "Cannot open logfile '$LOG_FILE': $!";

#if ($log) {
        #open(ULOG, ">>$log") or die "Cannot open userlog '$log' : $!";
#}

my $status;
my $mac;
my $time;
my $ip;

# default user
$user ||= $ENV{LOGNAME};

if ($host =~ /^$pat{mac}$/) {
        $mac = $host;
        logsay "using macaddr -host argument: $mac";
}
elsif ($host =~ /^$pat{ip}$/) {
        $ip = $host;
        logdie "not a DHCP addr: $ip" unless is_dhcp_ip($ip);
        $mac = dhcp_ip2mac($ip) or notify $user, "dhcp_jail_conf $action: can't resolve $host to
a macaddr";
        logsay "using -host argument: $ip (resolved to ether: $mac)";
}
elsif ($host =~ /^($pat{ip})@(.*)/) {
        $ip = $1;
        $time = $2;
        logdie "not a DHCP addr: $ip" unless is_dhcp_ip($ip);
        $mac = dhcp_ip2mac($ip, $time) or notify $user, "dhcp_jail_conf $action: can't resolve
$host to a macaddr";
        logsay "using -host argument: $ip@$time (resolved to ether: $mac)";
} else {
        logdie "bad host spec: $host\n";
}

logdie "Cannot modify secure-local!" if $mac eq '00:03:a0:88:d3:09';

$status = edit_dhcp_conf($DHCPD_CONF, $mac, $action, $OUTPUT_FILE, $ip, $user);



###############################################################################
#
# subs
#

#
# edit_dhcp_conf ( confg_file, macaddr, target_group )
#
# this edits the specified dhcpd config file, configuring  a host declaration
# for the macaddr (arg2) into the group specified by the third argument
#
# returns 1 if successful, and 0 if no changes were made
#
# Note that if the macaddr already has a host declaration of the type
# requested, no changes are made (return == 0).  If the macaddr already
# has a host declaration of a _different_ type (e.g. it is in the 'isolated'
# group and caller is requesting 'denyboot' group, the former will be
# removed and the latter added.

sub edit_dhcp_conf {

        my $config_file = shift;
        my $mac_c = shift;
        my $action = shift;
        my $output = shift || $config_file;
        my $ip = shift;
        my $user = shift || $ENV{LOGNAME};


        # make sure the config file exists
        -f $config_file or logdie "no such config file '$config_file'";
        # validate and cannonicalize the macaddr
        $mac_c  = fmt_mac($mac_c);
        my $mac = $mac_c;
        $mac =~ s/:/-/g;  # dash-separated version for host-decl-name

        # we write a tmp copy of the config file with revisions, later cp it over original
        my $config_tmp = "/tmp/dhcpd.conf.tmp";

        if ( -f $config_tmp && !$force ) {
                print STDERR "\nThe tmp config file '$config_tmp' already exists!\n";
                print STDERR "Either another version of this program is running, or it exited
improperly before.\n";
                print STDERR "Investigate, and/or remove the tmp file to continue.\n\n";
                exit 1;
        }

        # open our files
        open(CONF, $config_file) or logdie "Can't open dhcpd.conf file '$config_file': $!";
        logsay "successfuly opened (for reading) dhcpd.conf file '$config_file'";

        open(CONF_NEW, ">$config_tmp") or logdie "Can't open tmp dhcpd.conf file '$config_tmp':
$!";
        logsay "successfuly opened (for writing) temporary dhcp.donf output file '$config_tmp'";


        while (<CONF>) {

                # just duplicate every line until we reach a group delcaration
                unless ( /^group\s+(isolated|denyboot)\s+{/ ) {
                        print CONF_NEW;
                        next;
                }

                my $group = $1;  # name of the current group we're in

                print CONF_NEW;  # repeat the group starting line


                # now start a new, inner while loop to iterate over the interior
                # of the group declaration (ie inside the group {...} block).
                #
                # While on the inside of the group, only two lines are "special":
                #
                #   1) existing host declarations that match our MAC address
                #   2) end of the group, because we add new host declarations at the end
                #
                # everything else is just duplicated.

                while (<CONF>) {

                        if ( /^\s*host\s+([^-]+)-$mac/ ) {
                                # found a host declaration matching our macaddr

                                if ($action eq 'release') {
                                        # just ignore line (i.e. don't duplicate) and continue
                                        logsay "removing (explicit release) $mac_c from group
$group";
                                        <CONF>;
                                        next;
                                }
                                elsif ( $group eq $action ) {
                                        # the host is already in the file, in the same group as
target action
                                        logsay "$mac already in '$group' group";
                                        print CONF_NEW;
                                        # get following comment line
                                        $_ = <CONF>;
                                        print CONF_NEW;
                                        last;  # don't need to process this group any further
                                }
                                else {
                                        # host was found, but this is not the target group, so we
are
                                        # removing the the host declaration from this file by not
                                        # printing the line to the outputl ie. just continuing
                                        logsay "removing (in wrong group) $mac_c from group $group
at line $.";
                                        <CONF>;
                                        next;
                                }
                        }
                        elsif ( /^\s*}\s*$/ ) {
                                # at the end of group
                                # if this group is the target group, add the host declaration
                                if ( $action eq $group ) {
                                        printf CONF_NEW "\thost %s { hardware ethernet %s; }\n",
$group . '-' . $mac, $mac_c;
                                        printf CONF_NEW "\t# \@ %s ip: %s by: %s\n",
scalar(localtime(time)), $ip, $user;

                                        logsay "added $mac_c ($ip) to group $group at line $.";

                                        # also print result to stdout if not in verbose mode
                                        #printf "added $mac_c ($ip) to group $group at line $.\n"
if $verbose;
                                }

                                # and put the group-ending brace line
                                print CONF_NEW;
                                last;

                        }
                        else {
                                # not a "special" line in the group, just repeat it
                                print CONF_NEW;
                        }

                } # end of inner group while loop

        } # end input config file loop
        close CONF;
        close CONF_NEW;

        #
        my $status = system("diff -q $config_file $config_file > /dev/null");
        unless (system("cp $config_tmp $output")==0) {
                die "Error cp'ing $config_tmp to $output: $!";
        }
        unlink $config_tmp;

        return $status;
}



#
# dhcp_ip2mac ( ipaddr  [ timestamp ] )
#
# given an an ipaddr, possibly an optional timestamp (in localtime),
# get the corresponding macaddr from the leases file
#
sub dhcp_ip2mac {

        my $ip = shift;
        my $time = shift;

        die "bogus ip: $ip" unless $ip =~ /^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$/;

        if ($time and $time !~ /^(\d+|\d+\.\d+)$/ ) {

                if ($time =~ /(\d\d)-\d\d-\d\d-\d\d:\d\d:\d\d/)  {
                        if ($1 > 50) {
                                $time = "19$time";
                        } else {
                                $time = "20$time";
                        }
                }

                $time = str2time($time) or die "Can't convert '$time' into a unix time value";
        }

        open (F, "$LEASE_FILE") or die "Cannot open lease file '$LEASE_FILE': $!";

        my ($lip,$mac, $lease);

        local($/) = "}\n";

        my $i;
        print STDERR "reading leases      " if $debug;

        while (<F>) {

                $i++;
                printf STDERR "\b\b\b\b\b%5d", $i  if $debug;

                # only consider active leases
                next unless /binding state active/s;

                ($lip) = /lease (\S+)\s+{/s;  # lease ip

                next unless $lip eq $ip;

                if ($time) {
                        # extract starts and ends times from lease text
                        # the lease line is formatted like:
                        #       starts 3 2003/07/23 17:06:48;

                        my $s = (/starts \d ([^;]+);/s)[0];
                        my $e = (/ends \d ([^;]+);/s)[0];

                        # convert to unix time.  Note that lease times are GMT
                        # but our input time is expected to be local
                        my $starts = str2time( "$s GMT" );
                        my $ends = str2time( "$e GMT" );

                        next unless $time >= $starts && $time <= $ends;
                }

                # if we made it here, this lease pertains to the IP requested, so get mac
                ($mac) = /hardware ethernet ([^;]+);/s;
                $lease = $_;


        }
        close F;

        if ($mac) {
                logsay "ip $ip [$time] resolved to $mac";
                logsay "via lease:\n$_" if $debug;
        } else {
                logsay "couldn't resolve $ip [$time] resolved to a macaddr (not found in lease
file)";
                return undef;
        }

        return wantarray ? ($mac, $lease) : $mac;
}


#
# fmt_mac( mac )
#
# formats a macaddr to cannonical format (leading zero, lower case, colon-sep)
sub fmt_mac {
        my $mac = shift or die "no mac addr argument";
        my @n = split /:/, $mac;
        $mac = join(':', map { sprintf "%02s", lc($_) } @n);
        die "invalid macaddr '$mac'"
                unless $mac =~ /^[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}:[0-9a-f]{2}:[0-
9a-f]{2}$/;
        return $mac;
}


#
# utility function to grab mac and lease.
# this has no function in jailing
#sub resolve_ip {
        print "-->@ARGV\n";
        my ($mac, $lease) = dhcp_ip2mac(@ARGV);
        print "MAC address: $mac\n";
        print "Lease:\n";
        print "$lease\n";
        exit;
}



#
# usage()
#
sub usage {
print <<EOF;

Usage: $0 options macaddr [ macaddr ... ]

Options are:

  Required:
   --host             host to modify: macaddr | IP | IP\@timestamp

  Plus one of:
   --isolate          use isolated jail type for isolate, but not dangerous systems
   --deny-boot        use deny-booting jail type for dangerous systems
   --free             free host system from jail

  Utility options:
   --requestor        who requests
   --output           create new dhcpd.conf file ; leave input conf file untouched
   --conf             config file to edit [ default = $DHCPD_CONF ]
   --verbose          copy info that is logged to STDOUT

EOF
exit;
}



sub is_dhcp_ip {
        my $ip = shift or die "ip: $ip";
        my $shouldbe = join('.', reverse( split /\./, $ip)) . ".in-addr.dhcp.lbl.gov.";
        chomp(my $dig = `dig -x $ip +short`);

        return $dig =~ /^$shouldbe/ ? 1 : undef;
}


sub notify (@) {
        my $user = shift;
        my $message = shift;
        my $recipient = $EMAIL{$user} || $DEFAULT_EMAIL_NOTIFY;
        `/usr/bin/mail -s 'dhcp auto-edit failed' $recipient <<EOF
$message
EOF
`;

logsay "notified $user ($EMAIL{$user}) of failure; exiting";
exit;

}
```

dhcp_jailer_conf.pl

```
#!/usr/bin/perl

# $Id$

# this implements a simple batch-oriented wrapper around the
# dhcp_jail.pl script, to process a queue of jail requests
# in a single unit of work, within which the dhcpd.conf file is
# checked out, edited, checked in, and the server is restarted once.

# the purpose is to avoid ad-hoc editing of the dhcpd.conf file
# by multiple persons and possible thrashing of the server

# the queue is a directory of specially-named, empty files, where
# each file constitutes a single request.  The client program
# creates the file, and this program removes it once processed.

# this file format is a 3-field, plus (+) separated filename
# with request-type, host-spec, and requestor, eg:
#
#     isolate+00:00:e8:0d:63:e1+dedlow
#
# host spec can be ip and optional time
#
#     denyboot+131.243.92.243+dedlow
#     denyboot+131.243.92.243@23424324+dedlow
#     denyboot+131.243.92.243@03-09-15-00:00:00+dedlow


use strict;
use File::Basename;
use Getopt::Long;
require "dhcp_jailer.lib";

use vars qw($ERRFILE $LOGFILE $QUEUE_DIR $DHCP_CONF $DHCP_JAILER $DHCP_CONF_DIR
                       $program $verbose $restart $nocheckin);

# global vars
$QUEUE_DIR = "/home/pop/u0/capgrbell/dhcp-jail-requests";
$DHCP_CONF_DIR = "/tftpboot/etc";
$DHCP_CONF = "/tftpboot/etc/dhcpd.conf";
$DHCP_JAILER = "./dhcp_jailer_conf.pl";
$LOGFILE = '~ capgrbell/jailer.log';
$ERRFILE = '~capgrbell/jailer.err';
$program = basename($0, '.pl');

# temp default to no checkin
# $nocheckin = 1;

sub logsay (@);
sub logerr (@);
sub logdie (@);

#
# Main
#
GetOptions( 'verbose'         => \$verbose,
                       'restart'             => \$restart,
                       'no-check-in'  => \$nocheckin,
) or die;

open (LOG, ">>$LOGFILE") or die "Cannot open logfile '$LOGFILE': $!";
open (STDERR , ">>$ERRFILE") or die "Can't open error file '$ERRFILE': $!";


logsay "START $program ";

# sanity check
unless (-d $QUEUE_DIR && -w $QUEUE_DIR ) {
        logsay "Error: Queue dir '$QUEUE_DIR' doesn't exist or not writable";
        exit 1;
}


# check for work to do (presence of files in queue dir)
my @files = split /\s+/, `ls $QUEUE_DIR`;

if (@files == 0) {
        logsay "no files to process in queue dir $QUEUE_DIR";
        exit 0;
} else {
        logsay "%d files to process in queue dir $QUEUE_DIR", int @files;
}


# check out the RCS file
if (system("co -l -q $DHCP_CONF") == 0) {
        logsay "checked out dhcpd.conf file: '$DHCP_CONF'";
} else {
        logdie "check out of '$DHCP_CONF' failed: $!";
}


# call the jailer script for each file in the queue
for my $file (@files) {

        my ($action, $host, $who) = split /\+/, $file;
        die "invalid action '$action' : $file" unless $action =~
/^(denyboot|isolated|free|release)$/;

        $who ||= $ENV{LOGNAME};

        logsay "process queue file '$file'";

        #printf "$DHCP_JAILER -$action -requestor $who $host \n";


        if ( system("$DHCP_JAILER -$action -requestor $who $host") == 0 ) {

                logsay "successfuly ran $DHCP_JAILER -$action -requestor $who $host";
                unlink "$QUEUE_DIR/$file" or logdie "Failed to unlink request file
'$QUEUE_DIR/$file': $!";
                logsay "unlinked queue request file '$QUEUE_DIR/$file'";

        } else {

                logerr "Failed to run $DHCP_JAILER -$action -requestor $who $host";
                if (system ("mv $QUEUE_DIR/$file $QUEUE_DIR/.failed/$file") == 0 ) {
                        logsay "moved failed request to $QUEUE_DIR/.failed/$file";
                } else {
                        logdie "Can't move failed request to '$QUEUE_DIR/.failed/$file': $!";
                }
                #next;
                #cleanup();
        }



}


# validate modified dhcpd file
chdir $DHCP_CONF_DIR or logdie "cannot chdir to DHCP_CONF_DIR '$DHCP_CONF_DIR': $!";
if ( system("make lint >/dev/null 2>&1") == 0 ) {
        logsay "modified dhcpd conf file validates OK";
} else {
        logdie "modified dhcpd conf file fails to validate";
}



# check back in
if (!$nocheckin) {
        if ( system("echo '$0 auto edit' | ci -u -q $DHCP_CONF") == 0) {
                logsay "checkin of $DHCP_CONF OK";
        } else {
            logdie "failed check in: $!";
        }
} else {
        logsay "skipping RSC checkin because no-check-in flag was specified";
}


# restart dhcpd server, if requested
if ($restart) {
        chdir $DHCP_CONF_DIR or logdie "cannot chdir to DHCP_CONF_DIR '$DHCP_CONF_DIR': $!";
        if ( system("make restart >/dev/null 2>&1") == 0 ) {
                logsay "dhcpd restarted";
        } else {
                logdie "new dhcpd conf file failed to restart correctly";
        }
}



logsay "END $program ";



sub cleanup {
        if (system("co -q -f $DHCP_CONF") == 0) {
                logsay "abnormal exit, cleaning up; checking in partially processed dhcpd.conf";
        } else {
        logdie "Cannot check out RCS file: $!";
        }
        exit 1;
}


sub is_dhcp_ip {
        my $ip = shift ;
        my $shouldbe = join('.', reverse( split /\./, $ip)) . ".in-addr.dhcp.lbl.gov.";
        chomp(my $dig = `dig -x $ip +short`);

        return $dig eq $ shouldbe ? 1 : undef;
}
```